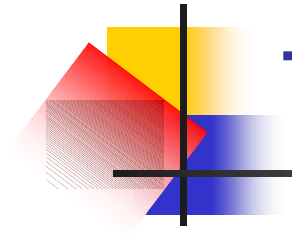
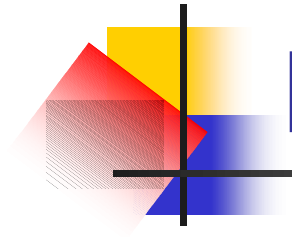


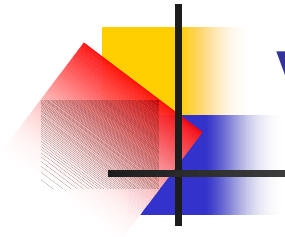
Tema 2. El lenguaje JAVA

- Nomenclatura habitual
- Variables
  - Tipos de variables
  - Tipos primitivos
  - Referencias
  - Arrays
- Operadores
  - Operadores de Java
  - Precedencia de operadores
- Sentencias de control
  - Sentencias condicionales
  - Sentencias de repetición
  - Sentencias break y continue



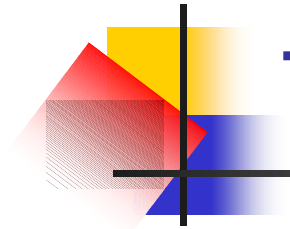
Nomenclatura habitual

- Los nombres de Java son sensibles a las letras mayúsculas y minúsculas
- Normas *recomendadas*:
  - Los nombres de **clases** e **interfaces** comenzarán siempre por mayúscula. Ej: HolaMundo.
  - Los nombres de **objetos**, **métodos** y **variables** empezarán por minúscula. Ej: main, x, y, ...
  - Los nombres de variables finales (constantes) se definirán siempre en mayúsculas. Ej: PI
  - Si un nombre consta de varias palabras, se pondrá en mayúscula la 1ª letra de la palabra que sigue a otra. Ej: HolaMundo, xCentro, yCentro,...



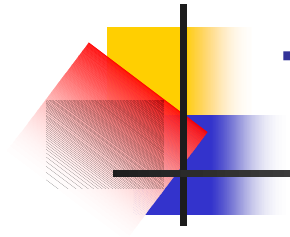
Variables

- Tipos de variables
  - Según la información que contienen:
    - Tipos primitivos: **char**, **byte**, **short**, **int**, **long**, **float**, **double** y **boolean**
    - Referencias: guardan la dirección de memoria donde se almacena un objeto o un array
  - Según su funcionalidad:
    - Variables miembro de una clase: se definen en una clase, fuera de cualquier método.
    - Variables locales: se definen dentro de un método o dentro de cualquier bloque entre llaves {}. Se crean dentro del bloque y se destruyen al finalizar el bloque.



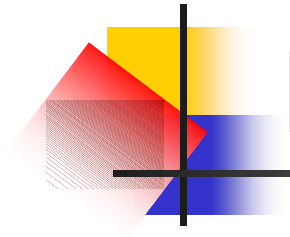
Tipos primitivos (i)

boolean	1 byte	true, false
char	2 bytes	Caracteres en código UNICODE
byte	1 byte	Entero entre –128 y 127
short	2 bytes	Entero entre –32.768 y 32.767
int	4 bytes	Entero entre –2.147.483.648 y 2.147.483.647
long	8 bytes	Entero entre –9.223.372.036.854.775.808 y 9.223.372.036.854.775.807
float	4 bytes	Real entre –3,402823E38 a –1,401298E-45 y de 1,401298E-45 a 3,402823E38
double	8 bytes	Real entre –1,797693E308 a –4,940656E-324 y de 4,940656E-324 a 1,797693E308



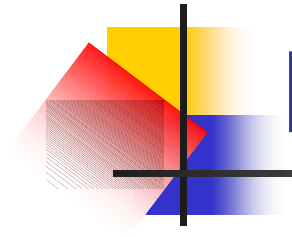
Tipos primitivos (ii)

- El tipo **boolean** no es un valor numérico
  - Sólo admite los valores **true** o **false**
  - No se identifica con el igual o distinto de 0, como en C/C++
- El tipo **char** contiene caracteres en código UNICODE
  - Incluye el código ASCII
  - Comprende los caracteres de casi todos los idiomas
- Los tipos de variables en Java están definidos en todas las plataformas (a diferencia de C/C++)
  - Ej: un **int** ocupa la misma memoria y tiene el mismo rango de valores en cualquier plataforma



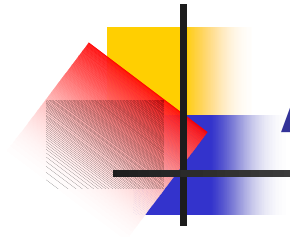
Declaración e inicialización

- Declaración:
  - **tipo nombreVariable;**
  - Ej: int x;
- Inicialización:
  - Puede especificarse un valor en la declaración.
    - Ej: int y = 10;
  - Si no se especifica nada:
    - Las variables de tipos primitivos se inicializan a 0 (excepto **boolean**, que se inicializa a **false**, y **char**, que se inicializa a **'\0'**)
    - Las variables de tipo referencia se inicializan a **null**.



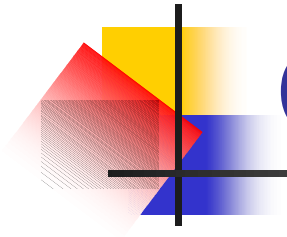
Referencias

- Una referencia es una variable que indica dónde está guardado un objeto en la memoria del ordenador
  - Java no permite acceder al valor de la dirección (se han eliminado los punteros)
- Al declarar una referencia no apunta a ningún objeto en particular, por lo que se le asigna el valor **null**.
  - Ejemplo: MiClase unaRef;
- Si se desea que la referencia apunte a un nuevo objeto, es necesario crear el objeto utilizando el operador **new**.
  - Ejemplo: MiClase otraRef = new MiClase();



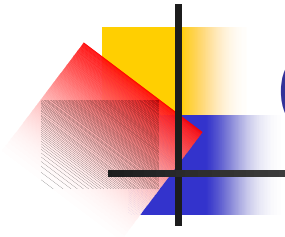
Arrays

- Un array o un vector es un tipo particular de referencia. Puede ser de variables primitivas o de objetos. En la declaración hay que incluir los corchetes **[]**.
- Ejemplos:
  - int [] vector; // Declaración de un array, inicializado a **null**
  - vector = new int [10]; // Vector de 10 enteros, inicializados a 0
  - double [] v = {1.23,5.45,3.45}; // Declaración e inicialización de un vector de 3 elementos
  - MiClase [] lista = new MiClase[5]; // Vector de 5 referencias a objetos. Las 5 referencias inicializadas a **null**
  - lista[0]=unaRef;



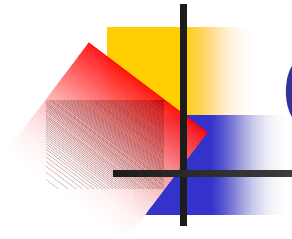
Operadores

- Aritméticos
- Incrementales
- Relacionales
- Lógicos
- A nivel de bits
- De asignación



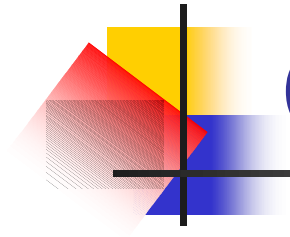
Operadores aritméticos

Operador	Uso	Descripción
+	op1 **+** op2	Suma op1 y op2
-	op1 **-** op2	Resta op2 de op1
*****	op1 ***** op2	Multiplica op1 y op2
/	op1 ***/*** op2	Divide op1 por op2
%	op1 **%** op2	Obtiene el resto de dividir op1 por op2



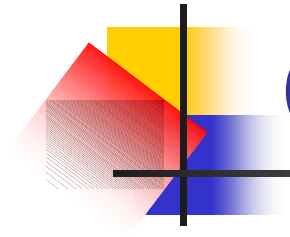
Operadores incrementales

Oper.	Uso	Descripción
++	op++	Incrementa op en 1 Evalúa el valor antes de incrementar
++	++op	Incrementa op en 1 Evalúa el valor después de incrementar
--	op--	Decrementa op en 1 Evalúa el valor antes de decrementar
--	--op	Decrementa op en 1 Evalúa el valor después de decrementar



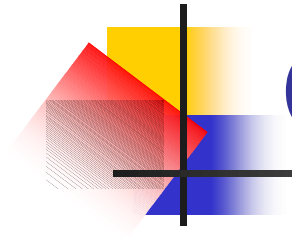
Operadores relacionales

Operador	Uso	Devuelve **true** si
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales **(no confundir con =)**
!=	op1 != op2	op1 y op2 son distintos



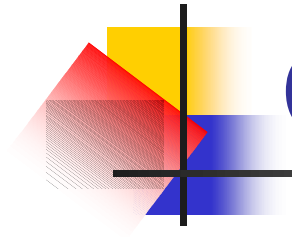
Operadores lógicos

Operador	Operación	Uso	Devuelve **true** si
&&	**and**	op1 && op2	op1 y op2 son **true**. Si op1 es **false**, no se evalúa op2
||	**or**	op1 || op2	op1 u op2 son **true**. Si op1 es **true**, no se evalúa op2
!	**not**	! op	op es **false**
&	**and**	op1 & op2	op1 y op2 son **true**. Siempre se evalúa op2
|	**or**	op1 | op2	op1 u op2 son **true**. Siempre se evalúa op2



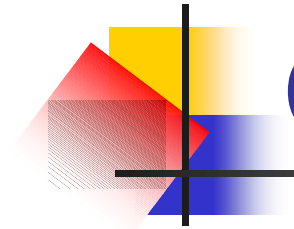
Operadores a nivel de bits (i)

Operador	Uso	Descripción
>>	op1 >> op2	desplaza a la derecha op2 bits de op1
<<	op1 << op2	desplaza a la izquierda op2 bits de op1
>>>	op1 >>> op2	desplaza a la derecha op2 bits de op1(sin signo)
&	op1 & op2	operador **and** a nivel de bits
|	op1 | op2	operador **or** a nivel de bits
^	op1 ^ op2	operador **xor** a nivel de bits (1 si sólo uno de los operandos es 1)
~	~ op	operador complemento (invierte el valor de cada bit)



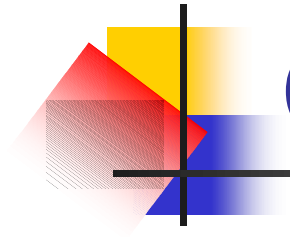
Operadores a nivel de bits (ii)

1. int a = 255, r = 0, m = 32;
2. r = a & 17; // r = 17
3. r = r | m; // r = 49
4. r = a & ~7; // r = 248
5. r = a >> 7; // r = 1
6. r = m << 1; // r = 64
7. r = m >> 1; // r = 16



Operadores de asignación

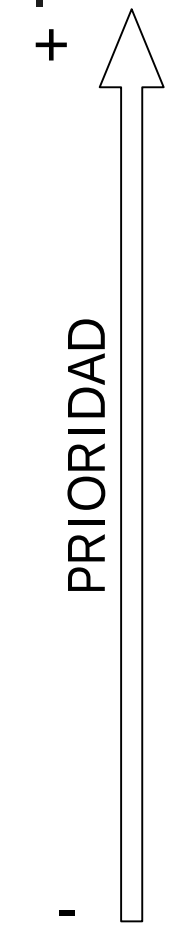
Operador	Uso	Equivale a
=	op1 = op2	
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2



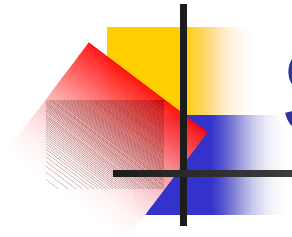
Otros operadores

- Operador condicional **?:**
  - *expresiónBooleana* ? op1 : op2
  - Devuelve op1 si *expresiónBooleana* es **true**, y op2 si es **false**.
  - Ejemplo:
    - double a=10.3, b=20.5, mayor=0;
    - mayor = (a>b) ? a : b; // mayor = 20.5;
- Operador **instanceof**
  - nombreObjeto instanceof NombreClase
  - Devuelve **true** si el objeto pertenece a la clase, y **false** si no pertenece

Precedencia de operadores

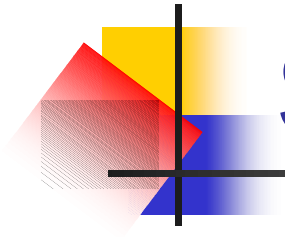


izda a dcha	( ) [ ] .
dcha a izda	- ~ ! ++ --
dcha a izda	new (*tipo*)*expr*
izda a dcha	* / %
izda a dcha	+ -
izda a dcha	<< >> >>>
izda a dcha	< <= > >= instanceof
izda a dcha	== !=
izda a dcha	&
izda a dcha	^
izda a dcha	|
izda a dcha	&&
izda a dcha	||
dcha a izda	? :
dcha a izda	= *= /= %= += -= <<= >>= >>>= &= |= ^=



Sentencias de control

- Sentencias condicionales
  - **if**
  - **if else**
  - **if elseif else**
  - **switch**
- Sentencias de repetición:
  - **while**
  - **do while**
  - **for**
- Sentencias **break** y **continue**



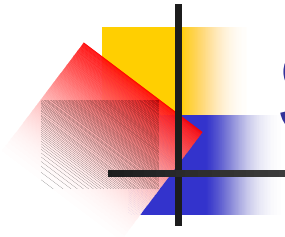
Sentencias condicionales (i)

- Sentencia **if**

```
if (expresiónBooleana) {
    // instrucciones si se cumple la condición
}
```

- Sentencia **if else**

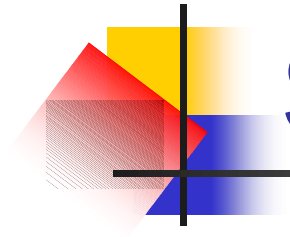
```
if (expresiónBooleana) {
    // instrucciones si se cumple la condición
}
else {
     // instrucciones si no se cumple la condición
}
```



Sentencias condicionales (ii)

- Sentencia **if elseif else**

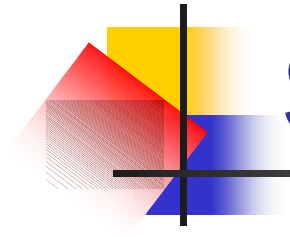
```
if (expresiónBooleana1) {
    // instrucciones
}
else if (expresiónBooleana2) {
    // instrucciones
}
else {
    // instrucciones
}
```



Sentencias condicionales (iii)

- Sentencia **switch**

```
switch (expresión) {
    case valor1:  // instrucciones si la expresión vale valor1
                  break;
    case valor2:  // instrucciones si la expresión vale valor2
                  break;
    case valor3:  // instrucciones si la expresión vale valor3
                  break;
    default: // instrucciones si la expresión vale
             // cualquier otro valor
}
```



Sentencias de repetición

- El bucle **while**

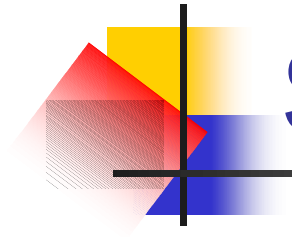
```
while (expresiónBooleana){
      sentencia; // se ejecuta mientras la expresión sea cierta
}
```

- El bucle **do while**

```
do {
   sentencia; // se ejecuta mientras la expresión sea cierta
}while (expresiónBooleana);
```

- El bucle **for**

```
for (inicialización; condición; incremento) {
   sentencia; // se ejecuta mientras la condición sea cierta
}
```



Sentencias **break** y **continue**

- Sentencia **break**
  - Finaliza inmediatamente la ejecución de una sentencia **switch** o de un bucle
- Sentencia **continue**
  - Finaliza la iteración que se esté ejecutando en un bucle, volviendo al comienzo del bucle y continuando en la siguiente iteración
  - Ejemplo:

```
for (int n=0;n<=100;n++){
   if(n%5!=0) continue;
   System.out.println(n  + " ");
}
```